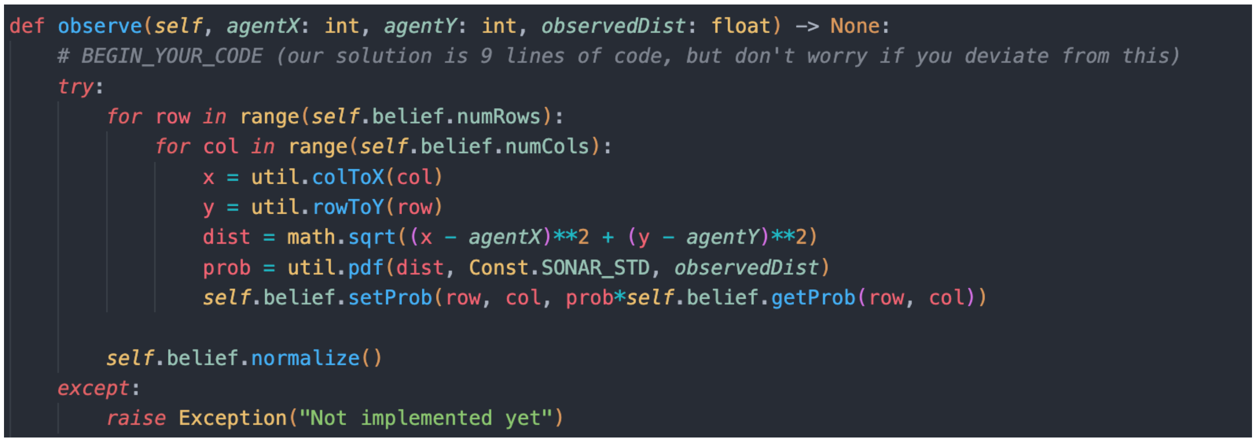
Homework 5: Car Tracking

Part I. Implementation (20%):

Part1



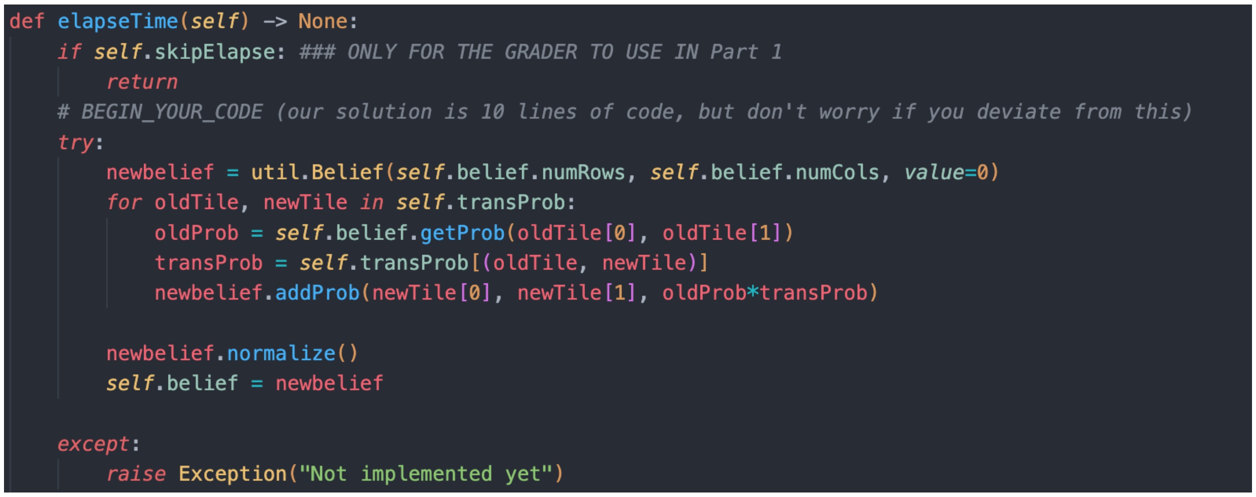
```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE (our solution is 9 lines of code, but don't worry if you deviate from this)
    try:
        for row in range(self.belief.numRows):
            for col in range(self.belief.numCols):
                x = util.colToX(col)
                y = util.rowToY(row)
                dist = math.sqrt((x - agentX)**2 + (y - agentY)**2)
                prob = util.pdf(dist, Const.SONAR_STD, observedDist)
                self.belief.setProb(row, col, prob*self.belief.getProb(row, col))

        self.belief.normalize()
    except:
        raise Exception("Not implemented yet")
```

Equation:

$$\mathbb{P}(H_t|E_1 = e_1, \cdots, E_t = e_t) \propto \mathbb{P}(H_t|E_1 = e_1, \cdots, E_{t-1} = e_{t-1})p(e_t|h_t)$$

In this part, we want to update the probability from our observations. From the equation above, we need to multiply the original probability by $p(e_t|h_t)$. From the specification, we know the observed distance is a Guassian random variable with mean equal to the true distance between our car and the other car. Therefore, we can get $p(e_t|h_t)$ by calculating PDF. After updating the probability, we normalize it to make its sum equal to one.

Part2



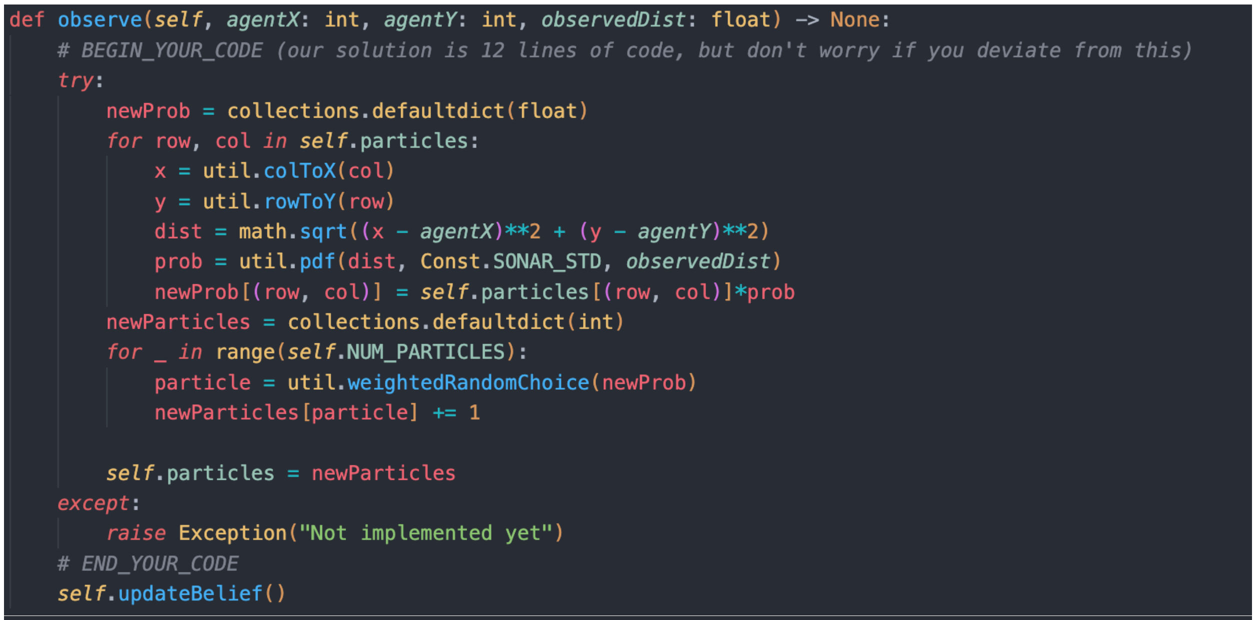
```python
def elapseTime(self) -> None:
    if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
        return
    # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
    try:
        newbelief = util.Belief(self.belief.numRows, self.belief.numCols, value=0)
        for oldTile, newTile in self.transProb:
            oldProb = self.belief.getProb(oldTile[0], oldTile[1])
            transProb = self.transProb[(oldTile, newTile)]
            newbelief.addProb(newTile[0], newTile[1], oldProb*transProb)

        newbelief.normalize()
        self.belief = newbelief

    except:
        raise Exception("Not implemented yet")
```

Equation:

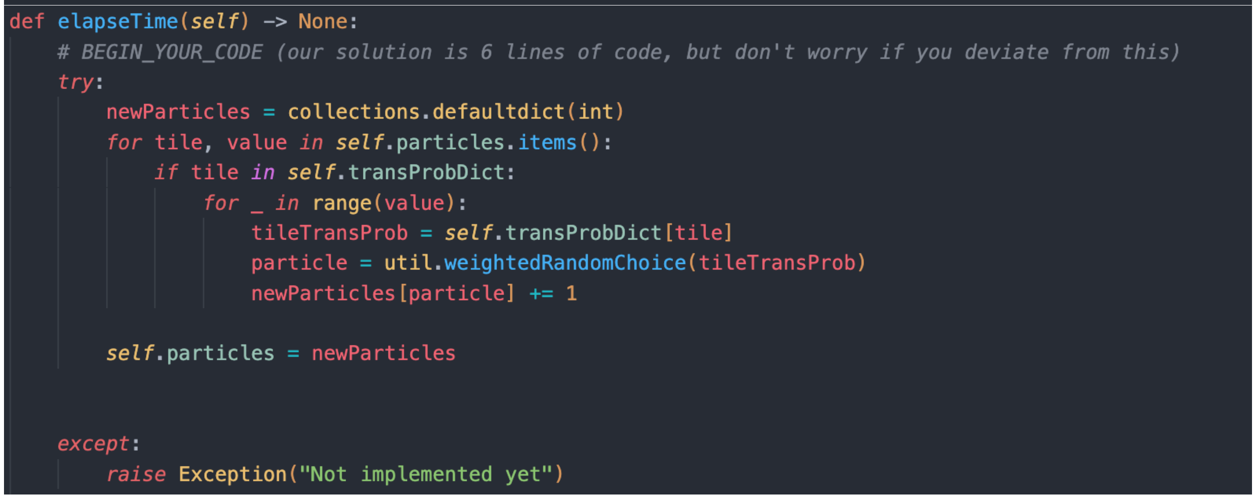
$$\mathbb{P}(H_{t+1} = h_{t+1} | E_1 = e_1, \cdots, E_t = e_t)$$
$$\propto \sum_{h_t} \mathbb{P}(H_t = h_t | E_1 = e_1, \cdots, E_t = e_t) p(h_{t+1} | h_t)$$

This part, we want to get the probability for the t+1st state with the previous t observations. We have got $P(Ht = ht|E_1 = e_1, \ldots, E_t = e_t)$ from part1. Therefore, we just need to multiply it with the transition probabilities. We construct a new belief with value equal to zero. Then, sum all the possible transition to the tiles respectively. In the end, we normalize it before replacing the belief with it.

Part3



```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE (our solution is 12 lines of code, but don't worry if you deviate from this)
    try:
        newProb = collections.defaultdict(float)
        for row, col in self.particles:
            x = util.colToX(col)
            y = util.rowToY(row)
            dist = math.sqrt((x - agentX)**2 + (y - agentY)**2)
            prob = util.pdf(dist, Const.SONAR_STD, observedDist)
            newProb[(row, col)] = self.particles[(row, col)]*prob
        newParticles = collections.defaultdict(int)
        for _ in range(self.NUM_PARTICLES):
            particle = util.weightedRandomChoice(newProb)
            newParticles[particle] += 1

        self.particles = newParticles
    except:
        raise Exception("Not implemented yet")
    # END_YOUR_CODE
    self.updateBelief()
```



```python
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
    try:
        newParticles = collections.defaultdict(int)
        for tile, value in self.particles.items():
            if tile in self.transProbDict:
                for _ in range(value):
                    tileTransProb = self.transProbDict[tile]
                    particle = util.weightedRandomChoice(tileTransProb)
                    newParticles[particle] += 1

        self.particles = newParticles


    except:
        raise Exception("Not implemented yet")
```

In this part, we do the same thing as part1 and part2. However, there is some positions that car has few possibilities to be there at each round. To avoid unnecessary computations, we use particle filtering. We use particles to stand for probabilities at each position. More the particles in the position, bigger the probabilities of car being there are. In the *observe* function, we need to update the probabilities with our observations. In this case, we update probabilities by multiplying the probability from *PDF* with the number of particles in that position because particles stand for probabilities. Then, we use *weightedRandomChoice* function to sample the new distribution of the particles with our updated probabilities. In the end, we use this new particle distribution to update our belief about where the car is.

In the *elapseTime*, we want to compute the transition probabilities. This time, we use *weightedRandomChoice* to sample the next position. If the number of particles is large, which means the high probabilities that car located in this position, we sample the next position many times. In the other words, this means that the destinations of these transitions have higher probabilities where the car is.

I want to mention the *if* statement to check if the tile is in the transition dictionary. I think this is unnecessary because if there is a tile not in the dictionary, it's impossible in the particles for the probabilities equal to zero. However, I have encountered this problem while I am debugging. I add this line just in case.

Part II. problems you meet and how you solve them:

This homework is easier than homework four as TA said. I think the most ideas the professor has mentioned during the class. I think we just need to figure out the relations between probabilities and the functions in the util.py, and we can finish this successfully. The part that I spend most time is part3. At first, I don't really get the point of particle filtering. I watch the video TA pasted in the specification twice and have some ideas about it. Then, I print some variables in the *ParticleFilter* to get the full understanding about this.